НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

«**До захисту допущено**»
В.о. завідувача кафедри
_____ __*О.А.Павлов*__
(підпис) (ініціали, прізвище)
“___” _____ 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки *6.050103 «Програмна інженерія»*

спеціальність *«Програмне забезпечення систем»*

на тему: *Програмні засоби для створення інтерактивного кіно*

Виконав: студент 4 курсу, групи ІП-52

Смілянець Федір Андрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *асистент Ісаченко Г.В.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц. к.т.н. Ліщук К.І.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент *старший викладач Виноградов Ю.М.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з трьох розділів, містить таблиць, додатків та 5 джерел – загалом 63 сторінок.

Об`єкт дослідження: веб-системи що використовуються для створення інтерактивного кіно з наявного відеоматеріалу та готових сценаріїв, з можливостями онлайн-перегляду та збереження прогресу, мова програмування для розмітки сценаріїв інтерактивного кіно.

Мета дипломного проекту: створити легко розширюваний веб застосунок, здатний до швидкого та зручного завантаження відеоматеріалу, розмітки переходів та перегляду.

У першому розділі було розроблено та імплементовано архітектуру монолітного веб застосунку та вбудованої в нього мови програмування. Побудовано структурну схему класів та діаграму послідовності.

У другому розділі проведено тестування монолітного веб застосунку за розробленим планом тестування. Описано процес тестування.

У третьому розділі описано розгортання та впровадження веб застосунку, а також наведено схему структурну розгортання.

У додатках наведено: опис програми, схема структурна класів програмного забезпечення, схема структурна послідовності виконання.

КЛЮЧОВІ СЛОВА: ІНТЕРАКТИВНЕ КІНО, ОНЛАЙН-КІНОТЕАТР, МОВА ПРОГРАМУВАННЯ

ABSTRACT

Explanatory note of the diploma project consists of 3 sections, annexes, tables and 5 sources – total 63 pages.

The object of study: web applications to employ in creation of interactive movies from existing video and scenarios, with possibility of watching published movies online. Programming language for interactive movie scenario markup.

The aim of the diploma project: create an easy extendable web application, capable of fast and easy video material upload, markup and view.

In the first section, the architecture of web application and programming language embedded into it was described and developed. A structural diagram of classes and a sequence diagram were constructed.

In the second secion, resulting web application was tested according to the developed test plan. The process of testing is described.

The third section describes the deployment and implementation of web application. The diagram of structural deployment is provided.

Annexes contain the description of the program, the diagram of the structural classes of the software, the scheme of the structural sequence of execution.

KEYWORDS: INTERACTIVE MOVIE, ONLINE CINEMA, PROGRAMMING LANGUAGE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Інтерактивний фільм – фільм, в якому глядач може приймати рішення що впливають на розвиток сюжету.

Блок – Цільна частина фільму, яка включає страйп, запитання після його відтворення та набір відповідей (серед яких може обирати глядач) з певними наслідками.

Наслідок – умовний перехід від одного блоку до іншого після прийняття рішення глядачем. Кожна відповідь може містити один чи декілька наслідків, один з яких є стандартним (настає якщо умови інших не справджуються), а інші можуть наставати в залежності від змінних та вже обраних відповідей. Також наслідок може змінити одну чи декілька змінних.

Страйп – Відеоролик, який є частиною цільного відеоматеріалу фільму.

Потокове відтворення – відтворення до повної прогрузки файлу.

Скінченний автомат (стейт-машина, finite state machine (FSM)) – абстракції, що використовується для описання шляху зміни стану об'єкта в залежності від поточного стану та інформації отриманої ззовні. Поняття скінченного автомата було запропоновано як математичну модель технічних приладів дискретної дії, оскільки будь-який такий пристрій (в силу скінченності своїх розмірів) може мати тільки скінченну кількість станів [1].

Стан перегляду – Блок (з страйпом, запитанням та відповіді на якому зараз знаходиться глядач), набір змінних та вже обраних відповідей.

Decision Carry-over, Відкладений вплив – Можливість прийнятих рішень впливати на події що стануться через декілька блоків через збереження прийнятих рішень та можливість використання їх в умовах наслідків.

Content Delivery Network (CDN) – географічно розподілена мережева інфраструктура, що дозволяє оптимізувати доправлення та розповсюдження контенту кінцевим користувачам в мережі Інтернет [2].

Google Cloud Storage – хмарне файлове сховище з опцією CDN [3].

ВСТУП

На сьогоднішній день велику популярність мають онлайн-сервіси з перегляду кіно. Найбільший з них – Netflix – за найскромнішими оцінками займає до 25-30% трафіку в США. Але навіть в ньому відсутня підтримка новітнього жанру кінотворчості – інтерактивного кіно. Найвідомішим прикладом є одна з серій серіалу «Чорне дзеркало», в якій глядач дійсно міг здійснювати вплив на хід сюжету відповідаючи на запитання.

Приголомшливий успіх цієї серії вказує на великий потенціал цього жанру водночас і як жанру кіно, і як своєрідного класу комп'ютерних ігор. Наразі найчастіше ігри та фільми такого плану збираються унікально, для кожного проекту програмне забезпечення будується з нуля.

Основною ціллю є створення системи для складання, публікування та перегляду інтерактивного кіно з хостингом відеоматеріалу на Google Cloud Storage.

Мета розробки – застосування практичних та теоретичних знань, отриманих студентом за час навчання для вирішення проблеми простого створення інтерактивного кіно.

Призначення результату розробки – запуск проекту як бізнесу, надання послуг публікації та перегляду інтерактивного кіно за допомогою розробленого веб-інтерфейсу.

Задачі:

- завантаження страйпів в систему;
- завантаження тексту програми в систему;
- лексинг, парсинг та генерація стейт-машини переходів між блоками;
- надання списку опублікованих фільмів;
- надання початкового стану перегляду;
- просування стану перегляду;

– надання страйпів для відображення.

Цілі:

– надати користувачеві (Авторові фільму чи глядачеві) інтерфейс для зручного створення та відтворення інтерактивного кіно;

– створити зручну та зрозумілу мову програмування інтерактивних фільмів.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Сучасні мережеві та програмні технології дозволяють без перебільшення мільйонам людей переглядати майже будь-які фільми, не виходячи з квартири чи дому і усе нове, що з’являється в цій сфері швидко знаходить підтримку. Однак, одна з можливостей такої сфери досі майже не випробовувалась. Наразі майже все кіно, яке взагалі можливо десь побачити є режисерським баченням якоїсь історії, прописаної чітко та послідовно сценаристами фільму. Такі фільми є цільними від початку і до кінця, а глядач є абсолютно пасивним і ніяк не впливає на те що відбувається в нього перед очима.

Ще в кінці 70х років минулого сторіччя комп’ютерні ентузіасти та експериментатори побачили пасивність читача книги, та створили жанр інтерактивної повісті, в якій читач може впливати на сюжет своїми рішеннями, або навіть прямо керувати одним або декількома героями [4].

На сьогодні інтерактивне кіно є диковинкою, до якої вдається вкрай мало митців через складність та унікальність реалізації механізму для кожного окремого фільму.

Програмний продукт для задоволення вимог має відповідати набору критеріїв:

– мова програмування має мати середовище для повноцінного запуску розробленої системи на сервері під керуванням операційних систем сімейства Linux;

– програмний продукт має надавати веб-сторінку у якості користувацього інтерфейсу;

– для підвищення інтерактивності веб-сторінка має бути виконана по принципу Single Page Application [5];

– виходячи з попереднього пункту програмний продукт має надавати API по протоколу HTTP, з кодуванням даних за допомогою існуючих текстових чи двійкових форматів;

– системою має бути передбачений значний відеотрафік, тому для збереження та видачі відео користувачам має бути використаний хмарний підхід у вигляді Google Cloud Storage чи іншої файлової CDN.

1.2 Змістовний опис і аналіз предметної області

Програмні засоби для створення інтерактивного кіно мають відповідати вимогам двух областей одночасно: інтерактивних історій та онлайн-кінотеатрів.

Як онлайн-кінотеатр засоби мають витримувати високе мережеве та хардварне навантаження стрімінгової передачі відеоконтенту, для чого має бути або реалізована власна система розповсюдження контенту, або використана готова. Такі системи мають підтримувати швидке та безпечне завантаження відеоконтенту з боку користувачів. Також завантаження відеоконтенту має проходити з мінімальною участю основного сервера бізнес-логіки для уникнення зайвого навантаження на мережевий канал.

Як засоби для створення інтерактивних історій, такі мають надавати простий та зручний інтерфейс для розробки та відлагодження схеми історії. Також такі засоби мають надавати функцію decision carry-over для спрощення організації довгострокових наслідків. Так як в інтерактивних історіях часто застосовуються різноманітні внутрішні параметри, як то відношення персонажів або різноманітні лічильники такі засоби мають надавати можливість створювати змінні, до читання та запису яких має бути наданий доступ в наслідках чи блоках.

1.3 Аналіз успішних ІТ-проектів

Інтерактивне кіно є порівняно новим жанром творчості, тому конкурентів у даного проекту не дуже багато. Найближчим аналогом може бути продукт під назвою «CtrlMovie» [6].

CtrlMovie – професійний інструмент для створення інтерактивного кіно для кінотеатрів, складається з декількох частей: програмного комплексу для створення фільму, аппаратного комплексу для установки в кінотеатрі та двох мобільних додатків для кіноглядачів – для Android та для iOS. Цей проект є досить серйозним конкурентом, оскільки для нього вже відзнято декілька фільмів які мали значний успіх на кінофестивалях та у кінотеатрах країн Європи. З іншої сторони, це є дуже складний та дорогий продукт, оскільки він максимально зав’язан на кінотеатри.

В CtrlMovie після кожного страйпу кінозалу відображається коротке запитання з двома відповідями, після чого кожен глядач обирає свою відповідь та фільм продовжує показ з тою відповіддю, яку відповіла більшість глядачів. З цієї точки зору проект, який є темою цього диплому, дозволяє більше ніж 2 відповіді.

Також, нічого не відомо про наявність прихованих змінних та відкладеного впливу в CtrlMovie.

Іншим ідейним конкурентом платформи що є метою цієї дипломної роботи є цілий жанр комп'ютерних ігор – інтерактивні повісті, тобто текстові, двомірні чи тримірні ігри, в яких гравець обмежено впливає на те, що відбувається на екрані приймаючи ті чи інакші рішення. Для цих ігор не існує готових каркасів, та для кожної гри всі механіки вибудовуються заново. Проект, який є темою цього диплому, може значно допомогти розробці таких ігор, оскільки включає в себе відокремлюваний проект Plotter, який саме є мовою програмування стейт-машин для інтерактивного кіно/повістей/т.д.

1.4 Розроблення функціональних вимог

Нижче наведено таблицю функціональних вимог. Структурну схему варіантів використання наведено в графічних матеріалах дипломної роботи.

Матрицю трассування наведено у таблиці 1.2.

Таблиця 1.1 – Функціональні вимоги та їх пріорітети

Варіант використання	Функціональна вимога	Пріорітет
Додавання та редагування фільмів	1) Програма має надавати можливість переглядати список створених користувачем фільмів 1.1) Програма має надавати можливість додавати фільми до списку 1.2) Програма має надавати можливість редагувати інформацію про фільми	Високий
Завантаження відеоконтенту	2) Програма повинна надавати список відеоконтенту фільму 2.1) Програма має надавати можливість додавати інформацію про відеоконтент 2.2) Програма має надавати можливість завантажувати відеоконтент до його інформації з комп’ютера користувача 2.3) Програма має надавати можливість редагувати інформацію про відеоконтент	Високий

Продовження таблиці 1.1

Варіант використання	Функціональна вимога	Пріорітет
Написання та відлагодження Plotter-коду	3) Програма має надавати можливість створювати та зберігати Plotter-код в інтерфейсі користувача 3.1) Програма має виконувати перевірку Plotter-коду на вірність.	Високий
Пошук фільмів для перегляду	1) Програма має надавати можливість переглядати список опублікованих користувачами фільмів 1.1) Програма має надавати можливість перейти до перегляду фільму зі списку	Високий
Перегляд відеоконтенту	2) Програма має надавати інтерфейс для перегляду відеоконтенту 2.1) Програма має надавати можливість здійснити сюжетне рішення після перегляду страйпу 2.2) Програма має надавати можливість розпочати перегляд з початку після завершення сюжетної дороги. 2.3) Програма має зберігати стан перегляду фільму для продовження на іншому комп’ютері.	Високий

Таблиця 1.2 – Матриця трассування

Функціональні вимоги / Варіант використання	Список фільмів	Інформація про фільми	Завантаження відео	Редагування Plotter-коду	Перевірка Plotter-коду	Список публічних фільмів	Перехід до фільму	Надання відеопотоку	Обробка переходів
Додавання та редагування фільмів	V	V							
Завантаження відеоконтенту			V						
Написання та відлагодження Plotter-коду				V	V				
Пошук фільмів для перегляду						V	V		
Перегляд відеоконтенту								V	V

1.5 Висновки по розділу

У цьому розділі було надано аналіз предметної області та конкурентів, а також порівняно наявні програмні засоби з існуючими реалізаціями схожих ідей і сформульовано функціональні вимоги.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Головним бізнес-процесом програмних засобів є бізнес-процес перегляду інтерактивного кіно, а саме обробка прийнятого глядачем рішення на сервері та надання наступного стану перегляду. BPMN-діаграму цього бізнес-процесу наведено в графічних матеріалах дипломного проекту.

При прийнятті рішення користувачем відправляється запит на сервер. На сервері здійснюється перевірка чи наявний в пам'яті екземпляр обробника станів перегляду для цього фільму. Якщо ні, то витягує з бази Plotter-текст фільму та створює обробник. В обробник передається поточний стан та прийняте рішення, після чого обробник знаходить потрібний наслідок виходячи з стану та прийнятого рішення, та повертає новий стан.

Після обробки стану перегляду новий стан зберігається в базі даних. Тоді з бази витягується інформація про відеофайл, відповідний поточному стану перегляду фільму користувачем, виконується запит на Google Cloud Storage для отримання тимчасового посилання на перегляд відео. Поточний стан у спрощеному вигляді (для закриття внутрішньої логіки фільму на сервері та унеможливлення чітингу, тобто обману серверу за допомогою підкладання несправжніх даних у стан) компонується з посиланням на перегляд відеоматеріалу та повертається на фронтенд. Фронтенд, реагуючи на відповідь серверу, завантажує нове відео та надає користувачу можливість його переглянути.

Іншим важливим бізнес-процесом цих програмних засобів є бізнес-процес програмування у мові Plotter.

Фронт-енд має надавати текстовий редактор, у якому має вестись розробка. На подію «Зберігання Plotter-коду», яка може бути викликана натисканням відповідної кнопки у користувацькому інтерфейсі або

натисканням комбінації клавіш «Ctrl-S» текст відправляється на сервер, де з нього створюється обробник станів перегляду, та якщо у коді були допущені помилки процедура створення обробників станів видасть відповідне повідомлення про синтаксичну, граматичну або семантичну помилку. Також якщо створення обробнику станів перевіряється чи всі використані у Plotter-коді страйпи дійсно існують та мають завантажений відеоматеріал, та у разі невідповідності коректному стану також повертається повідомлення про помилку яке має бути відображене користувачеві в інтерфейсі. Якщо за всю процедуру перевірки Plotter-коду не було видано жодних помилок код зберігається в базу, та існуючі обробники станів оновлюються. Також для уникнення конфліктів усі користувацькі стани перегляду скидаються в початкові.

2.2 Архітектура програмного забезпечення

Система призначена для розміщення на серверах з операційною системою Linux.

Система була повністю розроблена на мові програмування TypeScript.

Мова програмування Plotter, яка є частиною системи, була реалізована без застосування фреймворків, в модульному стилі, який дозволяє легке розширення функціоналу мови програмування додаванням відповідних лексем та коду для їх обробки. Вибір саме TypeScript надає можливість в подальшому застосовувати цю мову.. програмування в тому числі в браузерних застосунках на фронт-енд частині.

Серверну частину системи було розроблено за допомогою фреймворку Nest.JS та сховищ даних Redis [7] і MongoDB [8]. Використані паттерни включають DI [9] та MVC [10]. MVC дозволив відокремити технічний код надання API від бізнес-логіки та моделей даних, а DI допоміг зручно налаштувати організацію залежностей, сховавши ініціалізаційну логіку таким чином дозволяючи швидкі архітектурні зміни та зменшуючи об'єм можливих

структурних помилок. Результатом є простий, зрозумілий та легкорозширюваний серверний застосунок.

Браузерну частину системи було розроблено за допомогою фреймворку Angular. Було також знову застосовано паттерни DI та MVC, що дозволило побудувати прозорий процес менеджменту даних та роботи з сервером. Завдяки модульності архітектури та оптимізаційним налаштуванням браузерного застосунку на платформі Angular було досягнуто простоти розширення функціоналу застосунку, максимальної швидкодії та було знижено Time to Interaction до мінімально можливого рівня.

Інтерфейси описано у таблиці 2.1, класи та структури даних оглянуто у таблиці 2.2. Методи класів детально описано у таблиці 2.3.

Таблиця 2.1 – Опис інтерфейсів системи

Інтерфейс	Опис
Expression	Базовий тип виразів мови Plotter.
TreeNode	Базовий тип умовних операцій та операндів мови Plotter.
UserState	Інтерфейс середовища виконання мови Plotter що позначає стан перегляду.
Movie	Інтерфейс для роботи з сховищем документів типу Movie в базі даних.
Strip	Інтерфейс для роботи з субдокументами типу Strip що містяться в типі Movie.
User	Інтерфейс для роботи з сховищем документів типу в базі даних.

Таблиця 2.2 – Опис класів (структур) системи

Клас (структура)	Опис
AndExpression	Структура, яка позначає операцію І мови Plotter.
OrExpression	Структура, яка позначає операцію АБО мови Plotter.
NotExpression	Структура, яка позначає операцію НЕ мови Plotter.
EqualExpression	Структура, яка позначає операцію Дорівнює мови Plotter.
NotEqualExpression	Структура, яка позначає операцію НеДорівнює мови Plotter.

Продовження таблиці 2.2

Клас (структура)	Опис
LessThanExpression	Структура, яка позначає операцію МеншеНіж мови Plotter.
LessThanEqualExpression	Структура, яка позначає операцію МеншеАбоДорівнює мови Plotter.
BiggerThanExpression	Структура, яка позначає операцію Більше мови Plotter.
BiggerThanEqualExpression	Структура, яка позначає операцію БільшеАбоДорівнює мови Plotter.
Block	Структура, яка позначає оголошення блоку в мові Plotter.
Answer	Структура, яка позначає оголошення відповіді в мові Plotter.
Consequence	Структура, яка позначає оголошення наслідку в мові Plotter.
Effect	Структура, яка позначає операцію над змінною в наслідку в мові Plotter.
ImportVariablesStatement, ImportStripsStatement	Структури, які позначають оголошення імпортованих файлом мови Plotter змінних та страйпів.
Automata	Клас, призначений для парсингу даних скінченим автоматом.
Lexer	Клас, який розбиває файл мови Plotter на токени-лексеми.
Parser	Клас, який будує абстрактне синтаксичне дерево з файлу мови Plotter.

Продовження таблиці 2.2

Клас (структура)	Опис
SemanticChecker	Клас, який виконує семантичну перевірку абстрактного синтаксичного дерева.
Runner	Клас, який виконує логіку створювання та керування станами перегляду базуючись на файлі мові Plotter.
UserController	Клас, який організовує API роботи з користувачами.
MovieController	Клас, який організовує API роботи з фільмами.
MovieKeeperService	Клас, який надає доступ до стейт-машин створених фільмів та керує створенням та видаленням стейт-машин з пам’яті.
MovieService	Клас, який реалізує бізнес-логіку роботи з фільмами.
UserService	Клас, який реалізує бізнес-логіку роботи з користувачами.
StorageService	Клас, який керує взаємодією з Google Cloud Storage
AuthService	Клас, який виконує аутентифікацію дій користувачів

Продовження таблиці 2.2

Клас (структура)	Опис
BlocksComponent	Клас, який надає фронтенд для управління Plotter-текстом фільму
StripsComponent	Клас, який надає фронтенд для управління страйпами
EditorComponent	Клас, який надає фронтенд для управління фільмами користувача
HomeComponent	Клас, який надає домашню сторінку вебсайту
LoginComponent	Клас, який надає аутентифікаційний фронтенд
WatchComponent	Клас, який надає фронтенд для перегляду інтерактивного фільму
EditorService	Клас, який виконує передачу даних між модулем управління фільмами та бекендом
UserService	Клас, який виконує усю іншу передачу даних

Таблиця 2.3 – Опис методів класів та інтерфейсів системи

Клас/Інтерфейс	**Метод**	**Опис**
Lexer	AllTokens()	Парсить текст, котрим був ініціалізований лексер та повертає токени-лексеми
Lexer	nextToken()	Шукає наступний токен в рядку та повертає його. Пошук здійснюється за допомогою інших функцій
Lexer	skipWhitespacesAndNewLines()	Здвигає маркер поточного символу для пропуску пустих символів типу «Пробіл», «Перенос каретки» тощо.
Lexer	recognizeIdentifier()	Намагається розпізнати наступний токен починаючи з поточного символа як ідентифікатор, в разі помилки генерує виключення
Lexer	recognizeBrackets()	Парсить наступний токен як квадратну дужку і повертає його

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Lexer	recognizeNumber()	Намагається розпізнати наступний токен починаючи з поточного символа як число, в разі помилки генерує виключення
Lexer	recognizeString()	Намагається розпізнати наступний токен починаючи з поточного символа як строку, в разі помилки генерує виключення
Lexer	recognizeOperator()	Намагається розпізнати наступний токен починаючи з поточного символа як оператор, в разі помилки генерує виключення
Lexer	recognizeParenthesis()	Намагається розпізнати наступний токен починаючи з поточного символа як дужку, в разі помилки генерує виключення

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Lexer	recognizePunctuation()	Намагається розпізнати наступний токен починаючи з поточного символа як кому чи крапку з комою, в разі помилки генерує виключення
Lexer	recognizeBraces()	Намагається розпізнати наступний токен починаючи з поточного символа як квадратну дужку, в разі помилки генерує виключення
Parser	error(expectedType: string)	Формулює повідомлення про помилку в коді програми та кидає Exception
Parser	nextTokenWithCheck()	Переходить до наступного токена та виконує перевірку на тип
Parser	parseProgram()	Виконує побудову абстрактного синтаксичного дерева за текстом, переданим в конструктор

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Parser	parseImport()	Намагається розібрати вираз імпорту змінних чи страйпів починаючи з поточного токена, в разі проблеми генерує виключення
Parser	parseBlock()	Намагається розібрати вираз об'явлення блоку фільму починаючи з поточного токена, в разі проблеми генерує виключення
Parser	parseAnswer()	Намагається розібрати вираз позначення відповіді блоку починаючи з поточного токена, в разі проблеми генерує виключення
Parser	parseConsequence()	Намагається розібрати вираз позначення наслідку відповіді починаючи з поточного токена, в разі проблеми генерує виключення

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Parser	parseExp()	Намагається розібрати умовний вираз настання наслідку починаючи з поточного токена, в разі проблеми генерує виключення
Parser	parseStartingBlockDeclaration()	Намагається розібрати оголошення стартового блоку починаючи з поточного токена, в разі проблеми генерує виключення
SemanticChecker	check()	Виконує перевірку АСТ переданого в конструкторі, в разі знаходження помилки генерує виключення
SemanticChecker	createBlockNames()	Знаходить та зберігає назви блоків для подальшого використання в семантичній перевірці
SemanticChecker	checkImport()	Знаходить, перевіряє та зберігає імпортовані для подальшого використання в семантичній перевірці, знаходження помилки генерує виключення

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
SemanticChecker	checkBlocks()	Знаходить та перевіряє усі блоки на помилки, у разі знаходження генерує виключення
SemanticChecker	checkStartingDeclaration()	Знаходить та перевіряє вираз об’явлення стартового блоку, у разі помилки генерує виключення
PlotterStateManipulator	getFreshState()	Створює свіжий стан перегляду відповідно до Plotter-коду фільму та повертає його
PlotterStateManipulator	processDecision(state: UserState, answerName: string)	Знаходить наслідок за відповіддю, застосовує його до стану та повертає оновлений стан
PlotterStateManipulator	getConsequence(state: UserState, answer: Answer)	Перевіряє умови настання наслідків відповіді та обирає з них перший з справдженою умовою або стандартний

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
PlotterStateManipulator	checkConsequence(state: UserState, con: Consequence)	Перевіряє чи справджується умова наслідку в наданому стані
UserController	create(user: User)	Реєструє нового користувача
UserController	login(user: User)	Перевіряє логін та пароль користувача, створює токен, зберігає його та записує в cookies
UserController	logout()	Стирає токен з сховища Redis та cookies
UserController	me()	Повертає інформацію про користувача
MovieController	create(m: Movie, u: User)	Створює фільм та зберігає його в базі даних
MovieController	edit(m: Movie, u: User)	Зберігає зміни до фільму в базі даних
MovieController	saveText(mId: string, text: string, u: User)	Перевіряє Plotter-текст фільму на помилки та зберігає його в базі даних

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
MovieController	delete(m: Movie, u: User)	Знаходить фільм в базі даних, і якщо ініціатор дії є автором фільму видаляє фільм
MovieController	createStrip(m: Movie, stripName: string, u: User)	Знаходить фільм в базі даних, і якщо ініціатор дії є автором фільму створює страйп в фільмі
MovieController	publicMovies()	Повертає фільми що відкриті для перегляду авторами.
MovieController	restartMovie(mId: string, u: User)	Скидає стан перегляду фільму до початкового
MovieController	beginMovie(mId: string, u: User)	Повертає поточний стан фільму (для початку відтворення фільму після закриття вкладки з того моменту де користувач зупинився)
MovieController	progressMovie(mId: string, answerName: string, u: User)	Застосовує відповідь до стану перегляду та повертає новий стан
MovieController	getMovie(mId: string, u: User)	Повертає фільм якщо він публічний або ініціатор є автором цього фільму

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
MovieController	uploadStrip(mId: string, stripName: string, u: User)	Робить запит на Google Cloud Storage та повертає посилання на для завантаження страйпу страйпу
MovieController	uploadPic(mId, u: User)	Робить запит на Google Cloud Storage та повертає посилання на для завантаження обкладинки фільму.
MovieController	getByMe(u: User)	Повертає фільми, автором яких зазначено ініціатора запиту.
MovieKeeperService	createState(id: string)	Генерує та видає початковий стан перегляду для фільму за id.
MovieKeeperService	process (id: string, user: UserState, choice: string);	Просуває стан перегляду для фільму за id.
MovieKeeperService	initMachine(id: string)	Створює та запам’ятовує стейт-машину для фільму за id.
StorageService	getUploadLink(filename: string)	Генерує посилання для завантаження файлу на Google Cloud Storage

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
StorageService	getViewLink(filename: string)	Генерує посилання для перегляду файлу з Google Cloud Storage
UserService	create(user: User)	Створює нового користувача в базі даних
UserService	verify(user: User)	Перевіряє пароль користувача на правильність за допомогою алгоритму хешування bcrypt
UserService	saveState(user: User, movieid: string, state: UserState)	Зберігає стан перегляду фільму користувачем в базі даних
UserService	resetStates(movieid: string, freshState: UserState)	Застосовує оновлений початковий стан перегляду певного фільму до всіх користувачів в базі даних
MovieService	create(movie: Movie, user: User)	Створює новий фільм
MovieService	getByAuthorId(id: string)	Повертає усі фільми, створені певним користувачем
MovieService	createStrip(id: string, user: User, name: string, desc: string)	Додає новий страйп до фільму

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
MovieService	edit(id: string, edit: any, user: User)	Застосовує отримані зміни до фільму за відповідним ID
MovieService	saveText(id: string, text: string, user: User)	Перевіряє Plotter-текст фільму на коректність та зберігає його в базі даних
MovieService	uploadPic(id: string, user: User)	Посилання на завантаження обкладинки фільму на Google Cloud Storage
MovieService	uploadStrip(id: string, user: User, name: string)	Посилання на завантаження страйпу на Google Cloud Storage
MovieService	getPublic(skip?: number, limit?: number)	Повертає опубліковані авторами фільми, пагінований запит
MovieService	startMovie(user: User, movieid: string)	Повертає поточний стан перегляду користувачем фільму (або створює новий)

Продовження таблиці 2.3

MovieService	restartMovie(user: User, movieid: string)	Створює новий стан перегляду користувачем фільму.
MovieService	progressMovie(user: User, movieid: string, choice: string)	Просуває стан перегляду користувачем фільму
AuthService	saveUser(user: User, token: string)	Зберігає авторизаційний токен для користувача в сховище
AuthService	removeUser(token: string)	Видаляє авторизаційний токен користувача зі сховища
AuthService	validateUser(token: string)	Повертає користувача за авторизаційним токеном, якщо такий є в сховищі
BlocksComponent	saveText()	Пробує зберегти Plotter-текст фільму, та показує повідомлення про помилку в разі такої

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
EditMovieDialogComponent	save()	Зберігає інформацію про фільм в базі даних
StripsComponent	addStrip()	Створює новий страйп
StripsComponent	openUploadModal(i: string)	Відкриває модальне вікно для завантаження відеоматеріалу страйпу
UploadPicDialogComponent	save()	Завантажує обраний файл на Google Cloud Storage як обкладинку для фільму.
UploadVideoDialogComponent	save()	Завантажує обраний файл на Google Cloud Storage як страйп фільму.
EditorComponent	getMovies()	Завантажити фільми створені користувачем
EditorComponent	createMovie()	Відкриває модальне вікно створення фільму

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
EditorComponent	delete(mov: Movie)	Видаляє фільм з бази даних.
EditorComponent	editMovie(mov: Movie)	Відкриває модальне вікно редагування фільму
EditorComponent	uploadPic(mov: Movie)	Відкриває модальне вікно завантаження обкладинки фільму на Google Cloud Storage
EditorComponent	saveEdited(mov: Movie)	Зберігає оновлену інформацію про фільм на сервері
LoginDialog	register()	Реєструє нового користувача.
LoginDialog	login()	Авторизує користувача, отримує його авторизаційний токен.
WatchComponent	getMovieState()	Завантажує поточний стан перегляду відкритого фільму користувачем
WatchComponent	restart()	Скидає стан перегляду відкритого фільму користувачем

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
WatchComponent	progress(answerName: string)	Відправляє прийняте користувачем рішення на сервер та отримує новий стан перегляду користувачем фільму
WatchComponent	onPlayerReady(event: VgAPI)	Викликається при завантаженні відеоплеєра та зберігає інтерфейс для роботи з плеєром
UserService	getMovieState(movieid: string)	Завантажує поточний стан перегляду фільму
UserService	progressMovieState(movieid: string, choicename: string)	Відправляє прийняте рішення на сервер та отримує оновлений відповідно до рішення стан перегляду

Продовження таблиці 2.3

UserService	restartMovie(movieid: string)	Скидає стан перегляду фільму на початковий
UserService	getPublicMovies(limit: number, offset: number)	Завантажує опубліковані фільми
UserService	getUser()	Завантажує інформацію про поточного користувача
UserService	login(email: string, pass: string)	Авторизує користувача
UserService	register(name: string, pass: string, email: string)	Реєструє користувача
UserService	logout()	Скасовує авторизацію користувача

2.3 Конструювання програмного забезпечення

В програмному забезпеченні застосовано базу даних MongoDB та сховище Redis.

MongoDB містить два документи: «Користувач» та «Фільм», описані у таблицях 2.4 та 2.5 відповідно та один субдокумент «Страйп», описаний у таблиці 2.6.

Redis використовується для збереження аутентифікаційної інформації. При авторизації для користувача генерується токен, який записується в HTTP-only cookies запиту. В Redis використовуючи токен як ключ записується

ідентифікатор користувача, за рахунок чого виконується аутентифікація користувача за токеном.

Таблиця 2.4 – Схема документу «Користувач»

Поле	Тип	Опис
_id	ObjectID	Унікальний ідентифікатор
name	string	Ім’я, вказане користувачем
email	string	Електронна пошта користувача
hasMovieStates	string[]	ID фільмів, переглянутих користувачем
pass	string	Хеш та сіль паролю користувача
states	UserStates	Словник, в якому за id фільму знаходяться стани переглядів користувачем цих фільмів

Таблиця 2.5 – Схема документу «Фільм»

Поле	Тип	Опис
_id	ObjectID	Унікальний ідентифікатор
title	string	Назва фільму
desc	string	Пояснення до фільму
authorId	ObjectID	Ідентифікатор автора фільму
text	string	Plotter-код фільму
public	boolean	Флаг, який позначає доступність фільму до перегляду не автором.
strips	StripSchema	Страйпи фільму

Таблиця 2.6 – Схема документу «Страйп»

Поле	Тип	Опис
_id	ObjectID	Унікальний ідентифікатор
name	string	Назва страйпу
filename	string	Назва файлу, у якому зберігається відеоматеріал страйпу на Google Cloud Storage
desc	string	Пояснення до страйпу

На рисунку 2.1 наведено діаграму розгортання програмних засобів, та описано процес розгортання.

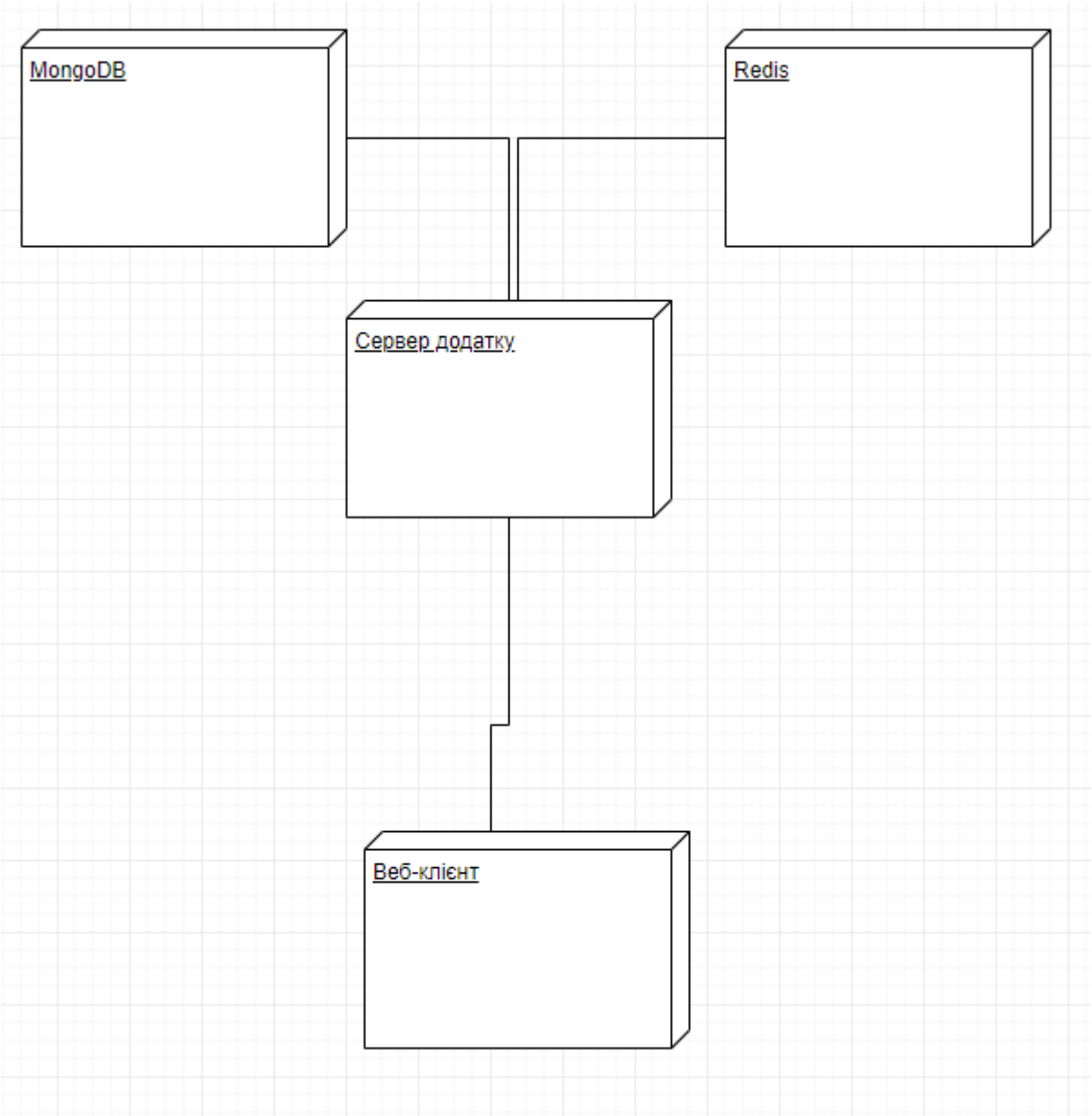

Рисунок 2.1 – Діаграма розгортання програмних засобів

Розгортання засобів відбувається у три етапи:

– Зовнішня підготовка;
– Внутрішня підготовка;
– Збірка проекту;

Зовнішня підготовка включає в себе інсталювання або перевірку підключення бази даних MongoDB та сховища Redis, налаштування операційної системи, встановлення середовища збірки та запуску проекту NodeJS + TypeScript, тощо.

Внутрішня підготовка засобів включає в себе установку проектних залежностей, фреймворків та бібліотек, тощо.

Збірка проекту включає в себе скачування та збірку фронт-енд проекту, їх правильна дислокація для сервінгу через серверну частину додатку, та інсталяції мови програмування Plotter, оскільки фронт-енд та Plotter знаходяться в окремих репозиторіях та потребують збірки перед запуском.

Фронт-енд засобів збирається з допомогою засобів полегшення загальної ваги клієнтського додатка, які включають стандартну мінімізацію через пошук контекстів та скорочення назв в цих контекстах, видалення зайвих символів, бандлинг, тощо. Також застосовані стандартні засоби полегшення загальної ваги клієнтського додатка включають утиліту інтелектуального зменшення бандлу програми за допомогою tree shaking, яка шукає використовуваний та відсікає зайвий програмний код. Бандлер повертає готовий набір, до якого можна доступитись зайшовши на index.html набору.

Запуск виконується після остаточної збірки та компіляції проекту в JavaScript або за допомогою ts-node.

2.4 Аналіз безпеки даних

Паролі користувачів зберігаються в хешованому вигляді (hash+salt), з застосуванням криптографічної хеш-функції bcrypt, що унеможливлює підбір пароля за прийнятний час.

Авторизаційні токени записуються сервером в HTTP-only cookies, що робить неможливим викрадення токену браузерними додатками, тощо. На сервері токени зберігаються в Redis.

Для взаємодії з Google Cloud використовується сервісний аккаунт, права якого легко обмежуються, та який легко закрити та замінити в разі зламу попереднього.

Для убезпечення сховища Google Cloud посилання на завантаження та посилання на перегляд мають обмежений час дійсності.

2.5 Висновки по розділу

У даному розділі було розроблено архітектуру інтерпретатора мови програмування Plotter, бекенду та фронтенду програмного коплексу та зазначено основні архітектурні патерни, обрано мову програмування та відповідні фреймворки. Було наведено детальний опис інтерфейсів, структур даних та класів, їх методів. Також було описано заходи з безпеки даних.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості програмного забезпечення

Аналіз якості розроблених програмних засобів є одним з ключових елементів процесу розробки та впровадження. Якість – суб’єктивне поняття, що позначає відповідність між очікуваною користувачем поведінкою та реальністю. Оскільки якість не є вимірюваною тестування як один з елементів контролю якості займається перевіркою методом порівняння чіткої специфікації з результатами роботи коду у кінцевому наборі тестів.

План аналізу якості програмного забезпечення наводить набір функцій програмних засобів які будуть протестовані, а також типи тестів, необхідні ресурси, тощо.

План аналізу якості програмного забезпечення включає виконання тестування всіх частей продукту, від інтерпретатора мови Plotter до авторизаційної частини.

У даному плані будуть протестовані наступні функції:

- Авторизація користувачів;
- Реєстрація користувачів;
- Створення фільмів;
- Додавання страйпів;
- Завантаження відеоматеріалу;
- Збереження Plotter-коду фільму;
- Перегляд списку фільмів;
- Перегляд відео;
- Прийняття рішень.

Тестові модулі:

- Авторизація користувача;
- Авторизація користувача з невалідним паролем;
- Реєстрація користувача;
- Реєстрація користувача з невалідним емейлом;
- Реєстрація користувача з невалідним паролем;
- Створення фільму;
- Видалення фільму;
- Публікування та депублікування фільму;
- Створення та видалення страйпів;
- Завантаження відео для страйпу;
- Завантаження Plotter-коду фільму;
- Завантаження коду, який використовує неіснючі страйпи;
- Завантаження коду з порушеннями синтаксису;
- Завантаження коду, який використовує неіснуючі змінні;
- Завантаження коду з іншими порушеннями семантики;
- Перегляд списку фільмів;
- Отримання стану перегляду та перегляд відео;
- Прийняття рішень та отримання нового відео;
- Рестарт фільму;

3.2 Підходи до тестування

Даний план включає використання наступних методів:

- Інтеграційного;

– Компонентного;

– Продуктивності;

3.2.1 Інтеграційне тестування

Метод інтеграційного тестування буде застосований до взаємодії між модулями системи:

– Робота API-сервера та Google Cloud Storage;

– Робота API-сервера та фронтенду;

– Робота API-сервера та мови Plotter;

3.2.2 Компонентне тестування

Метод компонентного тестування буде застосований до частин API-сервера та мови Plotter:

– Сервіси доступу до бази даних;

– Інтерпретація Plotter-коду та перемикання станів перегляду;

– Надавачі API;

3.2.3 Тестування продуктивності

Методом тестування продуктивності буде перевірена швидкодія надавачів API.

3.3 Критерії проходження тестування

3.3.1 Інтеграційне тестування, Компонентне тестування

Критерієм проходження інтеграційного та компонентного методів тестування критерій проходження є повне успішне виконання кожного пункту тесту. Якщо хоч один тест було виконано не успішно тестування вважається проваленим.

3.3.2 Тестування швидкодії

Критерієм проходження тестування швидкодії є повне успішне виконання кожного пункту тесту з усіма доступними наборами параметрів, як то об’єм запитів чи кількість одночасних запитів за допустимий час. В разі перевищення допустимого на виконання часу або невідповіді на запит взагалі тест вважається проваленим.

3.4 Процес тестування

3.4.1 Дані до тестів

Вхідними даними інтеграційного тесту є набори повідомлень, який отримує компонент системи ззовні від інших компонентів. Вихідними даними є результати роботи компоненту та його підлеглих.

Вхідними даними компонентного тесту є набори параметрів, контекстів та визначених результатів, які і є вихідними даними компонентного тесту.

Вхідними даними швидкодійного тесту є набори різноманітних даних які імітують натуральне використання програмного забезпечення. Вихідними даними являються швидкості обробки запитів, дані по навантаженню, тощо.

3.4.2 Задачі тесту

Задачею будь-якого тесту є перевірка правильності роботи програми в умовах виконання тесту, а також виявлення потенційних помилок у роботі.

3.5 Вимоги до середовища

3.5.1 Апаратна частина

Апаратна частина середовища виконання тестів має відповідати технічним вимогам запуску програмних засобів.

3.5.2 Вимоги до безпеки

Програмні засоби запущені для виконання тестування повинні працювати в окремій базі даних, а також мати окремий сервісний акаунт для доступу до окремого сховища Google Cloud Storage.

3.5.3 Інструменти

Інструментами для виконання перевірки програмних засобів є:

- Karma & Jasmine;
- Yandex Tank;
- Postman;

3.6 Опис контрольного прикладу

Таблиця 3.1 – Створення фільму

Мета тесту	Перевірка можливості створення фільму
Початковий стан	Відкриті програмні засоби
Вхідні дані	Назва та опис фільму
Схема проведення тесту	Перейти до сторінки «Editor», натиснути на кнопку «Create Movie», заповнити дані у форму, натиснути «Create».
Очікуваний результат	Сторінка оновить список фільмів та в ньому відобразиться новостворений фільм

Таблиця 3.2 – Завантаження відеоматеріалу страйпу

Мета тесту	Перевірка можливості завантаження страйпу
Початковий стан	Відкриті програмні засоби
Вхідні дані	Коректний відеоматеріал
Схема проведення тесту	Перейти до сторінки «Editor», обрати фільм, перейти на вкладку «Strips», створити страйп, натиснути «Upload Video, обрати файл, натиснути «Save».
Очікуваний результат	Сторінка завантажить обраний користувачем файл на Google Cloud Storage

Таблиця 3.3 – Редагування Plotter-тексту фільму

Мета тесту	Перевірка аналізатора мови Plotter
Початковий стан	Відкриті програмні засоби
Вхідні дані	Некоректний текст на мові програмування Plotter
Схема проведення тесту	Перейти до сторінки «Editor», обрати фільм, перейти на вкладку «Blocks», ввести дані, натиснути «Save».
Очікуваний результат	Сторінка відобразить помилковий стан та пояснення некоректності тексту.

Таблиця 3.4 – Перегляд фільму

Мета тесту	Перевірка обробника прийнятих користувачем рішень
Початковий стан	Відкриті програмні засоби
Вхідні дані	Заповнений та опублікований фільм
Схема проведення тесту	Обрати фільм серед опублікованих, продивитись страйп до кінця, обрати певне рішення
Очікуваний результат	Сторінка завантажить новий страйп у відпвідності до plotter-коду фільму, у кінці нового страйпу запитання та відповіді відповідатимуть plotter-коду фільму.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для повного розгортання даних програмних засобів необхідно виконати наступні задачі:

– створити Google Cloud проект, сегмент Google Cloud Storage та сервісний акаунт;

– встановити Redis;

– встановити MongoDB;

– встановити програмні засоби у вигляді серверу, фронтенду та мови Plotter.

4.1.1 Створення проекту Google Cloud

Для створення проекту Google Cloud Platform необхідно перейти на головну сторінку Google Cloud Platform (Рисунок 4.1), після чого натиснути на кнопку «Створити проект». Після заповнення форми (Рисунок 4.2) новий проект буде створено.

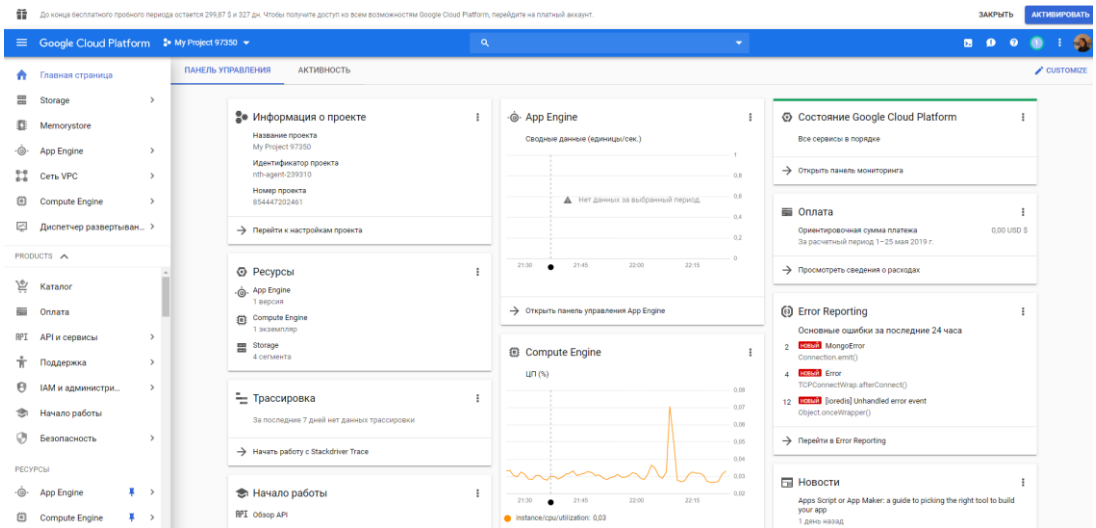

Рисунок 4.1 – Головна сторінка керування Google Cloud Platform

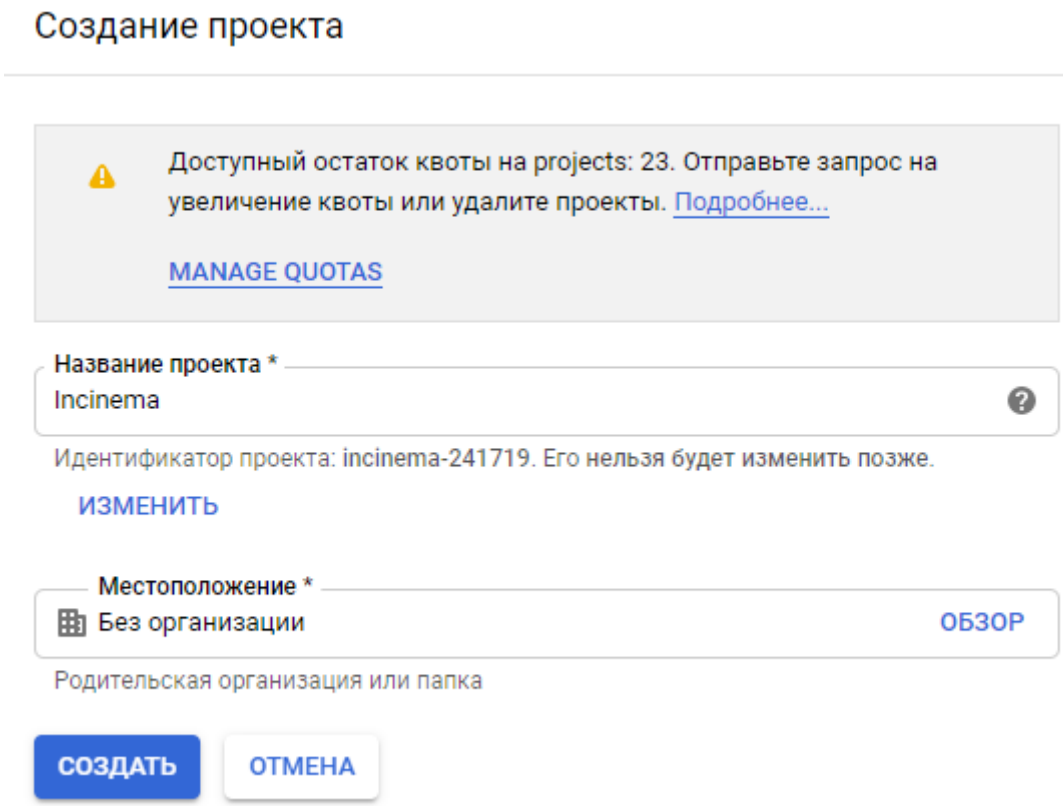

Рисунок 4.2 – Форма створення нового проекту

4.1.2 Створення сегменту Google Cloud Storage

Для створення нового сегменту потрібно перейти у меню Storage -> Огляд, та натиснути кнопку «Створти сегмент», після чого обрати клас сховища та його місцезнаходження у формі (Рисунок 4.3). Після натиснення кнопки «Створити», сегмент буде доступний до роботи.

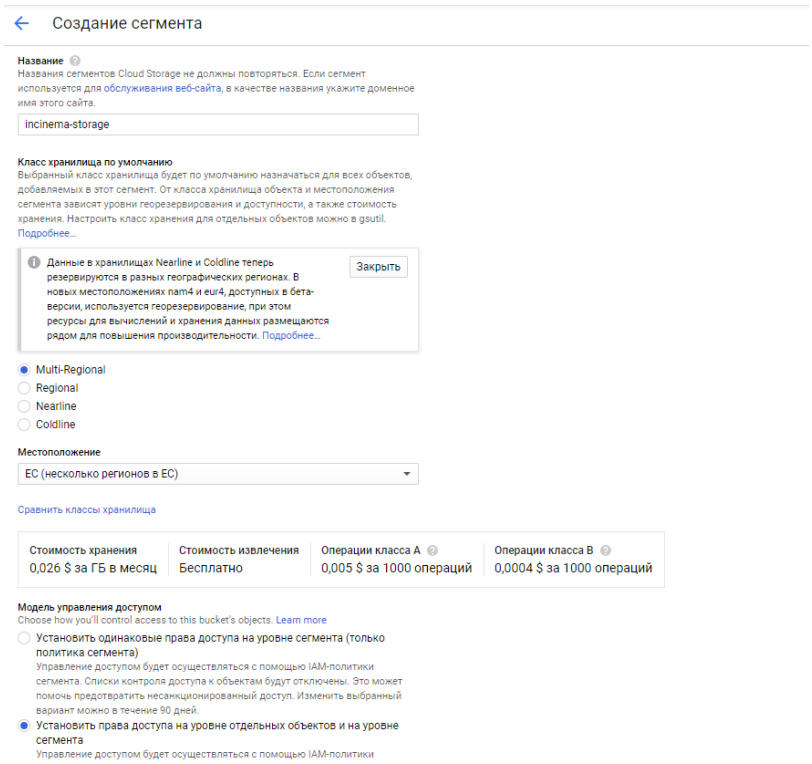

Рисунок 4.3 – Форма створення нового сегменту Google Cloud Storage

4.1.3 Створення сервісного акаунту Google Cloud

Для створення сервісного акаунту Google Cloud необхідно перейти на сторінку API та сервіси -> Облікові дані. У вкладці «Вікно запиту доступу OAuth» необхідно ввести назву додатку та зберегти зміни. Після чого потрібно натиснути кнопку «Створити облікові дані» у вкладці «Облікові дані», де обрати варіант «Ключ сервісного акаунта». У формі (Рисунок 4.4) необхідно створити новий сервісний акаунт обравши для нього відповідну політиці організації роль у графі «Сховище даних», та обравши метод збереження у JSON. Після натиснення кнопки «Створити» буде скачано файл, який потрібно зберегти для підстановки в серверну частину проекту.

Цей файл має зберігатись з уважністю до безпеки, оскільки цей файл містить аутентифікаційні дані до Google Storage та злодій, який може отримати такий файл у наслідок помилки програміста, сисадміна, DevOps-спеціаліста або іншого інженера може отримати за допомогою цього файлу доступ до усіх файлів проекту та деяких налаштувань.

Рекомендується зберігати такі файли, що містять аутентифікаційні дані, у шифрованих сховищах, та аутентифіковувати його використання для запуску серверної частини додатка іншими методами, для того щоб завадити випадковому протіканню файлу за межі мережі деплойменту програмних засобів в організації, яка їх використовує, займається їх підтримкою, тощо.

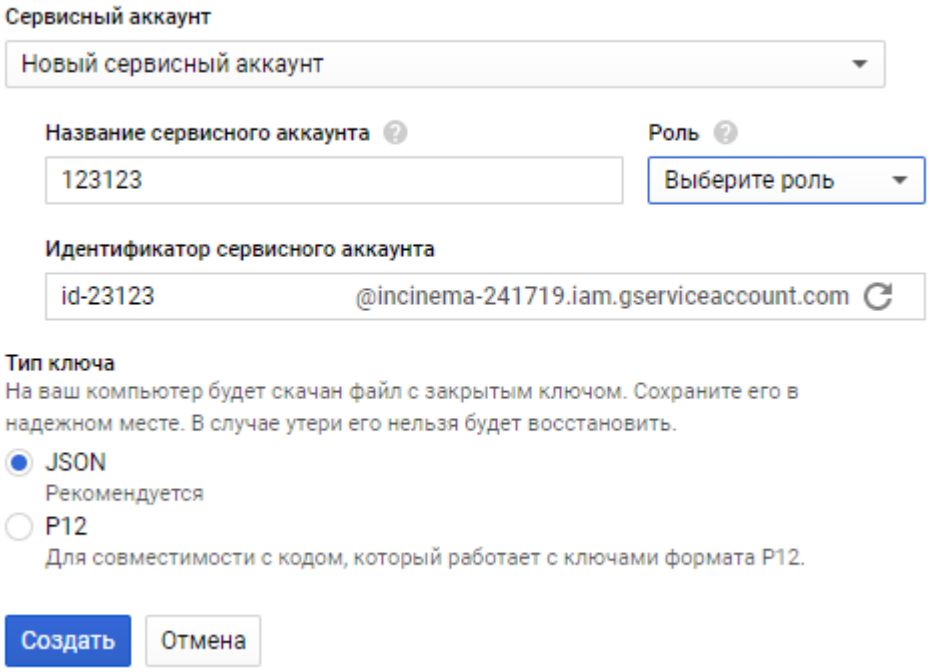

Рисунок 4.4 – Форма створення нового сервісного акаунта

4.1.4 Встановлення Redis

Для встановлення Redis необхідно перейти на офіційну сторінку виробника та слідувати описаним інструкціям.

4.1.5 Встановлення MongoDB

Для встановлення MongoDB необхідно перейти на офіційну сторінку виробника та слідувати описаним інструкціям.

4.1.6 Налаштування програмних засобів у вигляді серверу, фронтенду та мови програмування Plotter

Для налаштування програмних засобів потрібно перейти в папку з сирцевим кодом та виконати команди *npm i, npm i @angular/cli -g* та *./build-front.sh*. У корінь папки з сирцевим кодом необхідно помістити завантажений файл аутентифікації сервісного акаунта під ім'ям gcs.json та у файл app.module.ts ввести необхідні аутентифікаційні дані Redis та MongoDB. Після виконання вищезазначених дій програмні засоби можна буде запустити командою *npm start.*

4.2 Робота з програмним забезпеченням

4.2.1 Реєстрація

Для повноцінного користування програмними засобами користувач має зареєструватись на сторінці. Для цього потрібно натиснути на кнопку «Login/Register», у відкритому модальному вікні обрати варіант «Register», та заповнити форму (Рисунок 4.5).

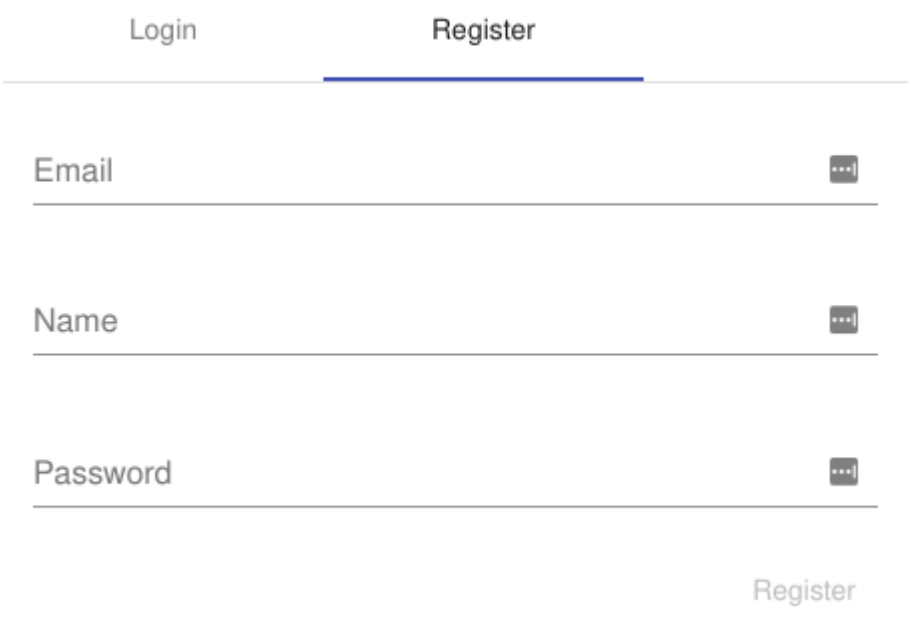

Рисунок 4.5 – Форма реєстрації

4.2.2 Вхід

Для початку роботи з програмними засобами потрібно увійти в систему, для чого в тому ж модальному вікні необхідно обрати варіант «Login» і заповнити відповідну сторінку, після чого головна сторінка засобів виглядатиме як на рисунку 4.6

Рисунок 4.6 – Головна сторінка сервісу після входу в систему

4.2.3 Створення фільму

Для створення фільму необхідно перейти у вкладку «Editor» та натиснути кнопку «Create Movie» (Рисунок 4.7), і заповнити відповідну форму (Рисунок 4.6). На цій самій сторінці для редагування даних фільму та його опублікування необхідно натиснути кнопку «Edit Details» на конкретному фільмі в списку.

Також на цій сторінці можливо завантажити постер фільму, який відображатиметься глядачам, для чого потрібно натиснути кнопку «Upload Poster», та завантажити файл у модальному вікні.

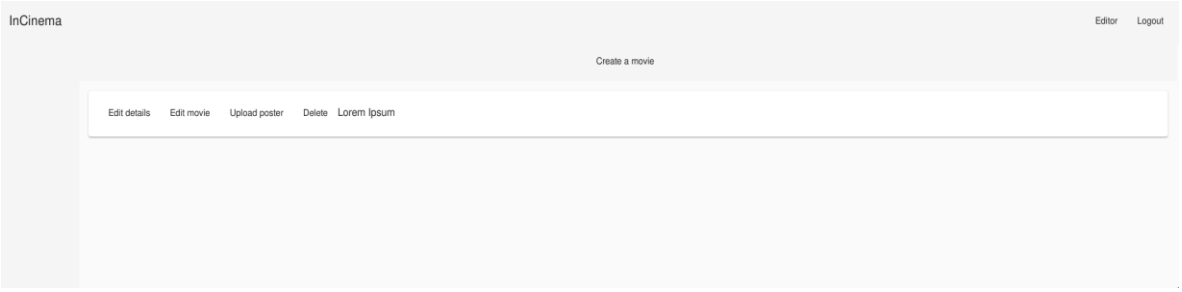

Рисунок 4.7 – Список фільмів

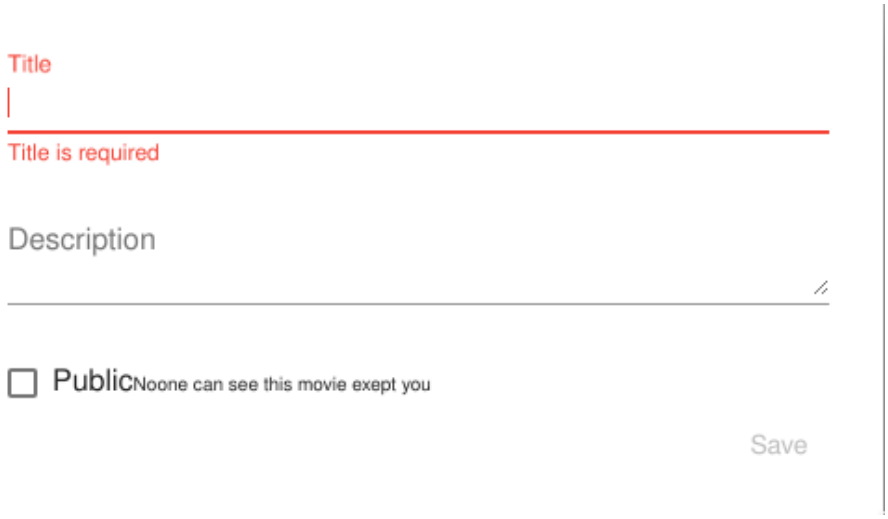

Рисунок 4.8 – Форма створення та редагування фільму

4.2.4 Створення та завантаження страйпів

У списку фільмів (Рисунок 4.7) необхідно натиснути кнопку «Edit Movie», після чого у меню зліва обрати пункт «Stripes». Відкриється список страйпів (Рисунок 4.9). Для створення страйпу необхідно ввести його ім’я (має бути унікальним) та коментар до страйпу, після чого з’явиться можливість завантажити страйп, натиснувши на кнопку «Upload video» навпроти відповідного страйпу.

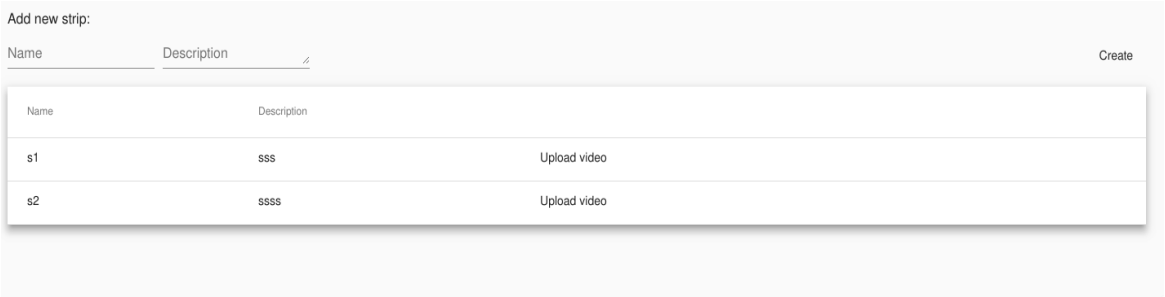

Рисунок 4.9 – Список страйпів

4.2.5 Редагування Plotter-тексту програми

У списку фільмів (Рисунок 4.7) необхідно натиснути кнопку «Edit Movie», після чого у меню зліва обрати пункт «Blocks». Відкриється текстовий редактор (Рисунок 4.10), у який можна вставити (або у якому писати) Plotter-програму, яка керуватиме сюжетним процесом перегляду користувача. Для збереження тексту необхідно натистнути на кнопку «Save text», або натиснути Ctrl+S. Поки що цей текстовий редактор є абсолютно базовим текстовим редактором без синтаксичного розмальовування шрифтів та інтелектуальних підказок, але зважаючи на API застосовуваного текстового редактора доповнення таким функціоналом можливо в майбутньому.

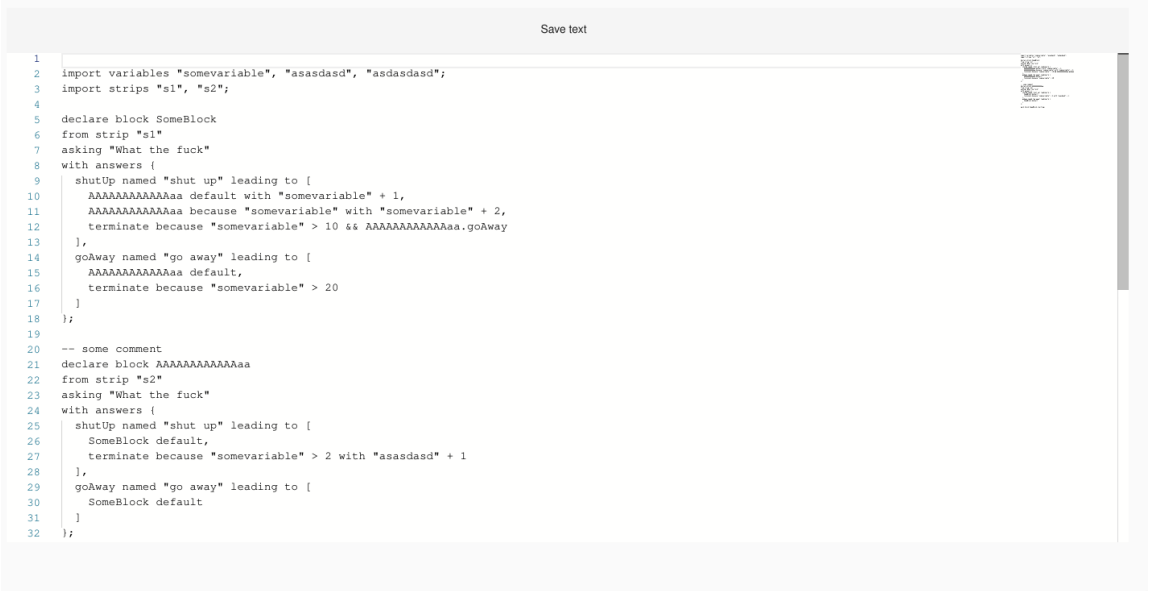

Рисунок 4.10 – Редактор мови програмування Plotter

4.2.6 Перегляд фільмів

Для перегляду фільму на головній сторінці необхідно обрати фільм, після чого відкриється сторінка перегляду фільму (Рисунок 4.10). Після закінчення перегляду страйпу з’явиться питання та набір відповідей (Рисунок 4.11), після натискання на одну з яких сторінка оновиться та з’явиться новий страйп. Якщо страйп є кінцевим у фільмі буде запропоновано розпочати перегляд фільму знову.

Рисунок 4.11 – Сторінка перегляду фільму

Рисунок 4.11 – Вибір відповіді на запитання після перегляду страйпу

4.3 Інструкція з програмування у мові Plotter.

Повноцінну інструкцію з програмування у мові Plotter наведено у документі «Інструкція програміста» дипломного проекту (КПІ.ІП-5218.045440.05.33).

ВИСНОВКИ

У даній дипломній роботі було проаналізовано предметну область, розроблено та втілено архітектуру програмних засобів. Після завершення процесу розробки було створено план тестування та протестовано програмні засоби у відповідності до нього.

Також було створено вичерпну інструкцію з підготовки середовища для деплою програмних засобів на апаратну чи віртуалізовану платформу, а також складено інструкції з використання, які описують як процес створення фільму, так і його перегляду.

В процесі роботи над цим дипломним проектом було застосовано, покращено та закріплено набір знань з баз даних, розробки специфічних мов програмування, архітектури програмного забезпечення, веб-розробки, тощо.

В результаті виконання роботи було розроблено повноцінний продукт, готовий до використання, але потребуючий розробки бізнес-моделі та доопрацювання під неї.

ПЕРЕЛІК ПОСИЛАНЬ

1) Wright, David R. (2005). "Finite State Machines" CSC215 Class Notes. David R. Wright website, N. Carolina State Univ.
2) Human Who Codes https://humanwhocodes.com/blog/2011/11/29/how-content-delivery-networks-cdns-work/ — How do Content Delivery Networks work
3) Google Cloud cloud.google.com — офіційна сторінка
4) Inform-Fiction https://www.inform-fiction.org/manual/html/s46.html — A short story of interactive fiction
5) Angular University https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/ — Angular Single Page Applications (SPA): What are the Benefits?
6) CtrlMovie http://www.ctrlmovie.com/ — офіційна сторінка
7) Redis redis.io — офіційна сторінка
8) MongoDB mongodb.com/nosql-explained — NoSQL Explained
9) Martin Fowler https://martinfowler.com/articles/injection.html — Inversion of Control Containers and the Dependency Injection pattern
10) Nick Heidke, Joline Morrison, and Mike Morrison, Department of Computer Science, University of Wisconsin-Eau Claire “Assessing the Effectiveness of the Model View Controller. Architecture for Creating Web Applications”

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду

Програмні засоби для створення інтерактивного кіно

(Найменування програми (документа))

DVD-R

(Вид носія даних)

23 арк, 753 Кб

(Обсяг програми (документа) , арк.,) Кб)

plotter/lexer.ts

```
import { Token, TokenTypes, CharUtils } from "./tokens";
import { NumberAutomata } from "./numAuto";

export class Lexer {
  private input: string;
  private position: number;
  private line: number;
  private column: number;

  constructor(input: string) {
    this.input = this.stripFinalNewline(input);
    this.column = 1;
    this.line = 1;
    this.position = 0;
  }

  private stripFinalNewline(input: string): string {
    const LF = typeof input === "string" ? "\n" : "\n".charCodeAt(0);
    const CR = typeof input === "string" ? "\r" : "\r".charCodeAt(0);
    if (input[input.length - 1] === LF) {
      input = input.slice(0, input.length - 1);
    }

    if (input[input.length - 1] === CR) {
      input = input.slice(0, input.length - 1);
    }
    return input;
  }

  public allTokens() {
    let token;
```

```
    let tokens: Token[] = [];

    do {
      token = this.nextToken();
      tokens.push(token);
      // console.log(token)
    } while (token.getType() !== TokenTypes.EOF);

    return tokens.filter(el => el.getType() !== TokenTypes.CommentStart);
  }

  private nextToken(): Token {
    if (this.position >= this.input.length) {
      return new Token(TokenTypes.EOF, "", this.line, this.column);
    }

    this.skipWhitespacesAndNewLines();

    const character = this.input.charAt(this.position);
    // console.log(`LOOKING FOR NEW TOKEN AT ${this.position} STARTING
WITH ${character}`)

    let token = null;
    if (!token && CharUtils.isLetter(character)) {
      token = this.recognizeIdentifier();
      if (!token) {
        throw new Error("Tokenizer error: unable to find identifier");
      }
    }

    if (!token && CharUtils.isDigit(character)) {
      token = this.recognizeNumber();
      if (!token) {
        throw new Error("Tokenizer error: unable to find number");
```

```
    }
  }

  if (!token && character === '"') {
    token = this.recognizeString();
    if (!token) {
      throw new Error("Tokenizer error: unable to find string");
    }
  }

  if (!token && CharUtils.isOperator(character)) {
    token = this.recognizeOperator();
    if (!token) {
      throw new Error("Tokenizer error: unable to find operator");
    }
  }

  if (!token && CharUtils.isParenthesis(character)) {
    token = this.recognizeParenthesis();
    if (!token) {
      throw new Error("Tokenizer error: unable to find parentheses");
    }
  }

  if (!token && CharUtils.isPunctuation(character)) {
    token = this.recognizePunctuation();
    if (!token) {
      throw new Error("Tokenizer error: unable to find punctuations");
    }
  }

  if (!token && CharUtils.isBrace(character)) {
    token = this.recognizeBraces();
    if (!token) {
```

```
        throw new Error("Tokenizer error: unable to find braces");
      }
    }
    if (!token && CharUtils.isBracket(character)) {
      token = this.recognizeBrackets();
      if (!token) {
        throw new Error("Tokenizer error: unable to find brackets");
      }
    }
    if (token) {
      // console.log(token)
      return token;
    } else {
      throw new Error(
        `Unrecognized character ${character} ${character.charCodeAt(
          0
        )} at line ${this.line} and column ${this.column}.
        See: ${this.input
          .split("")
          .slice(this.position - 10, this.position + 10)
          .join("")}`
      );
    }
  }

  skipWhitespacesAndNewLines() {
    while (
      this.position < this.input.length &&
      CharUtils.isWhitespaceOrNewLine(this.input.charAt(this.position))
    ) {
      if (CharUtils.isNewLine(this.input.charAt(this.position))) {
        this.line += 1;
        this.column = 1;
      } else {
```

```
      this.column += 1;
    }
    this.position += 1;
  }
}
skipToNextLine() {
  while (
    !CharUtils.isNewLine(this.input.charAt(this.position))
  ) {
    this.position += 1;
  }
  this.line += 1;
}

private recognizeParenthesis(): Token {
  let position = this.position;
  let line = this.line;
  let column = this.column;
  let character = this.input.charAt(position);

  this.position += 1;
  this.column += 1;

  if (character === "(") {
    return new Token(TokenTypes.Lparent, "(", line, column);
  }

  return new Token(TokenTypes.Rparent, ")", line, column);
}

private recognizeBrackets(): Token {
  let position = this.position;
  let line = this.line;
  let column = this.column;
```

```
    let character = this.input.charAt(position);

    this.position += 1;
    this.column += 1;

    if (character === "[") {
      return new Token(TokenTypes.Lbrack, "[", line, column);
    }

    return new Token(TokenTypes.Rbrack, "]", line, column);
  }

  private recognizePunctuation(): Token {
    let position = this.position;
    let line = this.line;
    let column = this.column;
    let character = this.input.charAt(position);

    this.position += 1;
    this.column += 1;

    if (character === ";") {
      return new Token(TokenTypes.Semi, ";", line, column);
    }

    return new Token(TokenTypes.Comma, ",", line, column);
  }

  private recognizeBraces(): Token {
    let position = this.position;
    let line = this.line;
    let column = this.column;
    let character = this.input.charAt(position);
```

```
    this.position += 1;
    this.column += 1;

    if (character === "{") {
      return new Token(TokenTypes.Lbrace, "{", line, column);
    }

    return new Token(TokenTypes.Rbrace, "}", line, column);
  }

  private recognizeOperator() {
    let character = this.input.charAt(this.position);

    if (CharUtils.isComparisonOperator(character)) {
      // console.log("comp op: ", character);
      return this.recognizeComparisonOperator();
    }

    if (CharUtils.isArithmeticOperator(character)) {
      return this.recognizeArithmeticOperator();
    }

    if (CharUtils.isLogicalOperator(character)) {
      return this.recognizeLogicalOperator();
    }

    // ...
  }

  private recognizeLogicalOperator() {
    let position = this.position;
    let line = this.line;
    let column = this.column;
    let character = this.input.charAt(position);
```

```
    // 'lookahead' is the next character in the input
    // or 'null' if 'character' was the last character.
    let lookahead =
      position + 1 < this.input.length ? this.input.charAt(position + 1)
: null;

    let isLookaheadAmpSymbol = lookahead !== null && lookahead === "&";
    let isLookaheadPipeSymbol = lookahead !== null && lookahead === "|";

    this.position += 1;
    this.column += 1;

    if (isLookaheadAmpSymbol || isLookaheadPipeSymbol) {
      this.position += 1;
      this.column += 1;
    } else {
      throw new Error(`Invalid token ${character}`);
    }

    switch (character) {
      case "&":
        return new Token(TokenTypes.And, "&&", line, column);
      case "|":
        return new Token(TokenTypes.Or, "||", line, column);
    }

    // ...
  }

  private recognizeComparisonOperator() {
    let position = this.position;
    let line = this.line;
    let column = this.column;
```

```
    let character = this.input.charAt(position);

    // 'lookahead' is the next character in the input
    // or 'null' if 'character' was the last character.
    let lookahead =
      position + 1 < this.input.length ? this.input.charAt(position + 1)
: null;

    // Whether the 'lookahead' character is the equal symbol '='.
    let isLookaheadEqualSymbol = lookahead !== null && lookahead === "=";

    this.position += 1;
    this.column += 1;

    if (isLookaheadEqualSymbol) {
      this.position += 1;
      this.column += 1;
    }

    switch (character) {
      case ">":
      return isLookaheadEqualSymbol
        ? new Token(TokenTypes.RtEq, ">=", line, column)
        : new Token(TokenTypes.Rt, ">", line, column);

      case "<":
      return isLookaheadEqualSymbol
        ? new Token(TokenTypes.LtEq, "<=", line, column)
        : new Token(TokenTypes.Lt, "<", line, column);

      case "=":
        return new Token(TokenTypes.Eq, "==", line, column);
      case "!":
        return isLookaheadEqualSymbol
```

```
            ? new Token(TokenTypes.Neq, "!=", line, column)
            : new Token(TokenTypes.Not, "!", line, column);
        default:
          break;
      }

      // ...
    }

    private recognizeArithmeticOperator() {
      let position = this.position;
      let line = this.line;
      let column = this.column;
      let character = this.input.charAt(position);

      // 'lookahead' is the next character in the input
      // or 'null' if 'character' was the last character.
      let lookahead =
        position + 1 < this.input.length ? this.input.charAt(position + 1)
  : null;

      // Whether the 'lookahead' character is the equal symbol '='.
      let isLookaheadPlusSymbol = lookahead !== null && lookahead === "+";
      let isLookaheadMinusSymbol = lookahead !== null && lookahead === "-";

      this.position += 1;
      this.column += 1;

      if (isLookaheadPlusSymbol || isLookaheadMinusSymbol) {
        this.position += 1;
        this.column += 1;
      }
      if (character === '-' && isLookaheadMinusSymbol) {
        this.skipToNextLine();
```

```
      return new Token(TokenTypes.CommentStart, '--', line, column);
    }

    switch (character) {
      case "+":
        return new Token(TokenTypes.Plus, "+", line, column);

      case "-":
        return new Token(TokenTypes.Minus, "-", line, column);

      case ".": // Not really an arithmetic op, but fit here so well
        return new Token(TokenTypes.Dot, ".", line, column);

      default:
        break;
    }
  }

  recognizeIdentifier() {
    let identifier = "";
    let line = this.line;
    let column = this.column;
    let position = this.position;

    while (position < this.input.length) {
      let character = this.input.charAt(position);

      if (
        !(
          CharUtils.isLetter(character) ||
          CharUtils.isDigit(character) ||
          character === "_"
        )
      ) {
```

```
      break;
    }

    identifier += character;
    position += 1;
  }

  this.position += identifier.length;
  this.column += identifier.length;

  if (CharUtils.isIdentifierReserved(identifier)) {
    return new Token(identifier, identifier, line, column);
  }

  return new Token(TokenTypes.Identifier, identifier, line, column);
}

private recognizeNumber() {
  let line = this.line;
  let column = this.column;

  // We delegate the building of the FSM to a helper method.
  let fsm = new NumberAutomata();

  // The input to the FSM will be all the characters from
  // the current position to the rest of the lexer's input.
  let fsmInput = this.input.substring(this.position);

  // Here, in addition of the FSM returning whether a number
  // has been recognized or not, it also returns the number
  // recognized in the 'number' variable. If no number has
  // been recognized, 'number' will be 'null'.
  let { recognized, value } = fsm.run(fsmInput);
  if (recognized) {
```

```
      this.position += value.length;
      this.column += value.length;

      return new Token(TokenTypes.IntegerLiteral, value, line, column);
    }

    // ...
  }

  recognizeString(): Token {
    let line = this.line;
    let column = this.column;
    const results = /^["|'][\w ]+["|']/.exec(
      this.input.substring(this.position)
    );
    if (!results) {
      return new Token(TokenTypes.Unknown, "", line, column);
    }
    this.position += results[0].length;
    this.column += results[0].length;
    return  new  Token(TokenTypes.StringLiteral,  results[0].substring(1,
results[0].length-1), line, column);
  }
}
```

plotter/parser.ts

```
import { TokenTypes, Token, CharUtils } from "./tokens";
import { Lexer } from "./lexer";

import {
  Statement,
  ImportStatement,
  ImportStripsStatement,
  ImportVariablesStatement,
  StartingBlockStatement,
```

```
  Block,
  Answer,
  Consequence,
  Expression,
  IntegerLiteral,
  NotExpression,
  AndExpression,
  VariableReference,
  OrExpression,
  EqualExpression,
  NotEqualExpression,
  LessThanExpression,
  GreaterThanExpression,
  LessThanEqualExpression,
  GreaterThanEqualExpression,
  Effect,
  AnswerReference
} from "./ast";
import { SemanticChecker } from "./semantics";

export class Parser {
  private binopLevels: any;
  private statements: Statement[];
  private lexer: Lexer;
  private tokens: Token[];
  private errorToken: Token;
  private currentToken: number;
  private errors: number;

  private get token() {
    return this.tokens[this.currentToken];
  }

  private initBinopLevels() {
```

```
    this.binopLevels = {};
    this.binopLevels[TokenTypes.And] = 10;
    this.binopLevels[TokenTypes.Or] = 10;
    this.binopLevels[TokenTypes.Lt] = 20;
    this.binopLevels[TokenTypes.Rt] = 20;
    this.binopLevels[TokenTypes.Eq] = 20;
    this.binopLevels[TokenTypes.Neq] = 20;
    this.binopLevels[TokenTypes.Plus] = 30;
    this.binopLevels[TokenTypes.Minus] = 30;
    this.binopLevels[TokenTypes.Dot] = 45;
    this.binopLevels[TokenTypes.Lbrace] = 50;
  }
  constructor(input: string) {
    this.initBinopLevels();
    this.lexer = new Lexer(input.trim());
    this.tokens = this.lexer.allTokens();
    this.currentToken = 0;
    this.errors = 0;
  }

  private error(expectedType: string) {
    const token = this.tokens[this.currentToken];
    if (token == this.errorToken) return;

    this.errorToken = token;
    this.errors++;
    throw new Error(
      `ERROR: ${token.getType()} at line ${token.getLine()}, column ${token.getColumn()};
Expected ${expectedType}`
    );
  }

  private eatToken(expectedType: string): boolean {
    const actualType = this.token.getType();
```

```
  if (expectedType === actualType) {
    this.nextToken();
    return true;
  } else {
    this.error(expectedType);
    return false;
  }
}

private eatTokenOf(expectedTypes: string[]): string {
  const actualType = this.token.getType();
  if (expectedTypes.includes(actualType)) {
    this.nextToken();
    return actualType;
  } else {
    this.error(expectedTypes.join(","));
    return actualType;
  }
}

private prevToken() {
  this.currentToken -= 1;
}
private nextToken() {
  this.currentToken += 1;
}
private nextTokenWithCheck(...expectedTypes: string[]) {
  // this.currentToken += 1;
  // const actualType = this.token.getType();
  // if (expectedType === actualType) {
  //   return true;
  // } else {
  //   this.error(expectedType);
  //   return false;
```

```
  // }
  this.currentToken += 1;
  const actualType = this.token.getType();
  if (expectedTypes.includes(actualType)) {
    return actualType;
  } else {
    this.error(expectedTypes.join(","));
    return actualType;
  }
}

private skipTo(follow: string[]) {
  while (this.token.getType() != TokenTypes.EOF) {
    for (let skip of follow) {
      if (this.token.getType() == skip) return;
    }
    this.nextToken();
  }
}

private rewind() {
  this.currentToken = 0;
}

public parseProgram() {
  this.statements = this.parseStatementList();

  const sem = new SemanticChecker(this.statements);
  sem.check();
  // this.eatToken(TokenTypes.EOF);
  return this.statements;
}

private parseStatementList(): Statement[] {
```

```
    this.skipTo([TokenTypes.Import]);
    const statementList: Statement[] = [this.parseImport(), this.parseImport()];
    while (this.token.getType() !== TokenTypes.Mark) {
      statementList.push(this.parseBlockDeclaration());
      // this.nextToken();
    }
    statementList.push(this.parseStartingBlockDeclaration());
    return statementList;
  }
  private parseBlockDeclaration() {
    const mtok = this.token;
    if (mtok.getType() !== TokenTypes.Declare) {
      this.error(TokenTypes.Declare);
    }
    const bl = new Block();
    this.nextTokenWithCheck(TokenTypes.Block);
    this.nextTokenWithCheck(TokenTypes.Identifier);
    bl.blockName = this.token.getValue();
    this.nextTokenWithCheck(TokenTypes.From);
    this.nextTokenWithCheck(TokenTypes.Strip);
    this.nextTokenWithCheck(TokenTypes.StringLiteral);
    bl.stripName = this.token.getValue();
    this.nextTokenWithCheck(TokenTypes.Asking);
    this.nextTokenWithCheck(TokenTypes.StringLiteral);
    bl.question = this.token.getValue();
    this.nextTokenWithCheck(TokenTypes.With);
    this.nextTokenWithCheck(TokenTypes.Answers);
    this.nextTokenWithCheck(TokenTypes.Lbrace);
    while (this.token.getType() !== TokenTypes.Rbrace) {
      bl.answers.push(this.parseAnswer());
    }
    this.nextTokenWithCheck(TokenTypes.Semi);
    this.nextToken();
    // console.log('FINISHED BLOCK')
```

```
  return bl;
 }
 private parseAnswer(): Answer {
  this.skipTo([TokenTypes.Identifier]);
  this.currentToken -= 1;
  const answ = new Answer();
  this.nextTokenWithCheck(TokenTypes.Identifier);
  answ.name = this.token.getValue();
  this.nextTokenWithCheck(TokenTypes.Named);
  this.nextTokenWithCheck(TokenTypes.StringLiteral);
  answ.text = this.token.getValue();
  this.nextTokenWithCheck(TokenTypes.Leading);
  this.nextTokenWithCheck(TokenTypes.To);
  this.nextTokenWithCheck(TokenTypes.Lbrack);
  while (this.token.getType() !== TokenTypes.Rbrack) {
   answ.consequences.push(this.parseConsequence());
  }
  this.nextToken();
  // console.log('FINISHED ANSWER')

  return answ;
 }
 private parseConsequence(): Consequence {
  const cons = new Consequence();
  this.nextTokenWithCheck(TokenTypes.Identifier, TokenTypes.Terminate);
  if (this.token.getType() == TokenTypes.Terminate) {
   cons.isTerminating = true;
  } else {
   cons.nextBlock = this.token.getValue();
  }
  this.nextTokenWithCheck(TokenTypes.Default, TokenTypes.Because);
  if (this.token.getType() == TokenTypes.Default) {
   cons.isDefault = true;
```

```
  } else {
    this.nextToken();
    cons.condition = this.parseExp();
    this.prevToken();
  }
  this.nextTokenWithCheck(TokenTypes.With, TokenTypes.Comma, TokenTypes.Rbrack);
  if (this.token.getType() == TokenTypes.With) {
    this.nextTokenWithCheck(TokenTypes.StringLiteral);
    cons.effect = new Effect();
    cons.effect.variable = this.token.getValue();
    this.nextTokenWithCheck(TokenTypes.Plus, TokenTypes.Minus);
    cons.effect.operation = this.token.getValue();
    this.nextTokenWithCheck(TokenTypes.IntegerLiteral);
    cons.effect.amount = Number(this.token.getValue());
    this.nextToken();
  }

  // console.log('FINISHED CONSEQUENCE')



  return cons;
}

private parseExp(): Expression {
  const lhs = this.parsePrimaryExp();
  return this.parseBinopRHS(0, lhs); // check for binops following exp
}

private parsePrimaryExp(): Expression {
  switch (this.token.getType()) {
    case TokenTypes.IntegerLiteral:
      const intValue = Number(this.token.getValue());
      this.eatToken(TokenTypes.IntegerLiteral);
      return new IntegerLiteral(intValue);
```

```
    case TokenTypes.StringLiteral:
      const stringVal = this.token.getValue();
      this.eatToken(TokenTypes.StringLiteral);
      return new VariableReference(stringVal);

    case TokenTypes.Identifier:
      const ar = new AnswerReference();
      ar.blockName = this.token.getValue();
      this.eatToken(TokenTypes.Identifier);
      this.eatToken(TokenTypes.Dot);
      ar.answerName = this.token.getValue();
      this.eatToken(TokenTypes.Identifier);
      return ar;

    case TokenTypes.Not:
      this.eatToken(TokenTypes.Not);
      return new NotExpression(this.parseExp());

    case TokenTypes.Lparent:
      this.eatToken(TokenTypes.Lparent);
      const exp = this.parseExp();
      this.eatToken(TokenTypes.Rparent);
      return exp;

    default:
      // unrecognizable expression
      this.eatToken(TokenTypes.Unknown);
      this.nextToken();
      return null;
  }
}

private parseBinopRHS(level: number, lhs: Expression): Expression {
```

```
// continuously parse exp until a lower order operator comes up
while (true) {
  // grab operator precedence (-1 for non-operator token)
  let val = this.binopLevels[this.token.getType()];
  const tokenLevel = val !== undefined ? val : -1;

  // either op precedence is lower than prev op or token is not an op
  if (tokenLevel < level) return lhs;

  // save binop before parsing rhs of exp
  const binop = this.token.getType();
  this.eatToken(binop);

  let rhs = this.parsePrimaryExp(); // parse rhs of exp

  // grab operator precedence (-1 for non-operator token)
  val = this.binopLevels[this.token.getType()];
  const nextLevel = val !== undefined ? val : -1;

  // if next op has higher precedence than prev op, make recursive call
  if (tokenLevel < nextLevel) rhs = this.parseBinopRHS(tokenLevel + 1, rhs);

  // build AST for exp
  switch (binop) {
    case TokenTypes.And:
      lhs = new AndExpression(lhs, rhs);
      break;
    case TokenTypes.Or:
      lhs = new OrExpression(lhs, rhs);
      break;
    case TokenTypes.Eq:
      lhs = new EqualExpression(lhs, rhs);
      break;
    case TokenTypes.Neq:
```

```
        lhs = new NotEqualExpression(lhs, rhs);
        break;
      case TokenTypes.Lt:
        lhs = new LessThanExpression(lhs, rhs);
        break;
      case TokenTypes.Rt:
        lhs = new GreaterThanExpression(lhs, rhs);
        break;
      case TokenTypes.LtEq:
        lhs = new LessThanEqualExpression(lhs, rhs);
        break;
      case TokenTypes.RtEq:
        lhs = new GreaterThanEqualExpression(lhs, rhs);
        break;
      default:
        this.eatToken(TokenTypes.Unknown);
        break;
    }
  }
}
private parseStartingBlockDeclaration() {
  const mtok = this.token;
  if (mtok.getType() !== TokenTypes.Mark) {
    this.error(TokenTypes.Mark);
  }
  this.nextTokenWithCheck(TokenTypes.Block);
  this.nextTokenWithCheck(TokenTypes.Identifier);
  const blockname = this.token;
  this.nextTokenWithCheck(TokenTypes.Starting);
  this.nextTokenWithCheck(TokenTypes.Semi);
  const st = new StartingBlockStatement();
  st.blockName = blockname.getValue();
  return st;
}
```

```
  private parseImport() {
    const importToken = this.token;
    if (importToken.getType() !== TokenTypes.Import) {
      this.error(TokenTypes.Import);
    }
    let importType = this.nextTokenWithCheck(
      TokenTypes.Strips,
      TokenTypes.Variables
    );

    const importstatement: ImportStatement =
      importType === TokenTypes.Strips
        ? new ImportStripsStatement()
        : new ImportVariablesStatement();

    while (this.token.getType() !== TokenTypes.Semi) {
      this.nextTokenWithCheck(TokenTypes.StringLiteral);
      importstatement.pushElement(this.token.getValue());
      this.nextTokenWithCheck(TokenTypes.Comma, TokenTypes.Semi);
    }
    this.nextToken();

    return importstatement;
  }
}
```